Stochastic Identification and Clustering of Malware with Dynamic Traces

Curtis B. Storlie, Scott Vander Wiel, Daniel Quist, Blake Anderson, Curtis Hash, Nathan Brown

Los Alamos National Laboratory

Date: March 26, 2012

Abstract

A novel approach to malware classification is introduced based on analysis of instruction traces that are collected dynamically from the program in question. The method can be applied on-line in a sandbox environment, and is intended for eventual host-based use, provided the engineering hurdle of sampling the instructions executed by a given process without disruption to the user can be satisfactorily addressed. The procedure represents an instruction trace using a Markov Chain structure in which the transition matrix, P, has rows modeled as Dirichlet vectors. The malware class (malicious or benign) is modeled using logistic regression with variable selection on the elements of P, which are observed with error. The utility of the method is illustrated on a sample of traces from malware and non-malware programs, and the results are compared to other leading detection schemes (both signature and classification based). Finally, a novel clustering approach is presented based on a probabilistic change similarity measure. This approach is effective at identifying previously examined malware which is similar to a newly identified instance, to aid in reverse engineering.

Keywords: Malware Detection; Classification; Clustering; Elastic Net; Relaxed Lasso; Adaptive Lasso; Logistic Regression; Splines

Running title: Malware Detection

Corresponding Author: Curtis Storlie, `storlie@lanl.gov`

1 Introduction

Malware (short for malicious software) is a term used to describe a variety of forms of hostile, intrusive, or annoying software or program code. It was recently estimated that one in four computers operating in the US are infected with malware (Organisation for Economic Co-operation and Development 2008). More than 286 million unique variants of malware were detected in 2010 alone (Symantec 2011), and it is widely believed that the release rate of malicious software is now far exceeding that of legitimate software applications (Symantec 2008). A large majority of the new malware is created through simple modifications to existing malicious programs or by adding some code obfuscation techniques such as a *packer* (Royal, Halpin, Dagon, Edmonds & Lee 2006). A packer compresses a program much the same way a compressor like Pkzip does, then attaches its own decryption/loading stub which 'unpacks' the program before resuming execution normally at the program's original entry point (OEP).

Malicious software is growing at such a rate that commercial antivirus vendors (AV) are not able to adequately keep up with new variants. In the recent study, AV Comparatives' proactive retrospective tests (Antivirus Comparatives 2011), detection was found to be substantially less than the ideals touted by AV company product literature. There are two methods for antivirus scanners to implement their technology. The first is via a static signature scanning method, which uses a sequence of known bytes in a program and tests for the existence. The second is using generic or heuristic detection technologies. These heuristics are what we seek to specifically improve upon in this paper. Unfortunately, even though most of the new malware is very similar to known malware, it will often not be detected by signature-based antivirus programs (Christodorescu & Jha 2003, Perdisci, Dagon, Fogla & Sharif 2006), until the malware signature eventually works its way into the database, which can take weeks or even longer. AV Comparatives' retrospective tests demonstrate the effectiveness of the AV industry's accuracy at finding previously unknown threats. The AV software is updated on a predetermined date T1. At a future date T2, a month later, the AV software is then used to scan threats that have appeared after T1. The test removes the ability to develop static signatures, and provides a good test of the heuristic defenses of the AV software. The highest detection rate among the 12 AV programs considered in the study is 67%, although this AV software also had what was described as a large number (25) false alarms (the false detection *rate*, however, is unknown as the number of benign programs used in the study is not reported).

Because of the signature based susceptibility to new malware, classification procedures based on statistical and machine learning techniques have been employed with varied success, to make a decision about the integrity of an unknown program. These methods have generally revolved around n-gram analysis of the static binary or dynamic trace of the malicious program (Reddy & Pujari 2006, Reddy, Dash & Pujari 2006, Stolfo, Wang & Li 2007, Dai, Guha & Lee 2009). These methods have shown great promise in detecting zero-day malware, but there are drawbacks related to these approaches. The two parameters generally associated with n-gram models are n, the length of the subsequences being analyzed, and L, the number of n-grams to analyze. For larger values of n and L, there is a much more expressive feature space that should be able to discriminate between malware and benign software more easily. But with these larger values of n and L, one runs into the *curse of dimensionality*: the feature space becomes too large and there is not have enough data to sufficiently condition the model. With smaller values of n and L, the feature space is too small and discriminatory power is lost.

The data sources used to classify programs include binary files (Kolter & Maloof 2006, Reddy et al. 2006, Stolfo, Wang & Li 2005), binary disassembled files (Bilar 2007, Shankarapani, Ramamoorthy, Movva & Mukkamala 2010), dynamic system call traces (Bayer, Moser, Kruegel

& Kirda 2006, Hofmeyr, Forrest & Somayaji 1998, Rieck, Trinius, Willems & Holz 2011), and most recently dynamic instruction traces (Anderson, Quist, Neil, Storlie & Lane 2011, Dai et al. 2009). Although some success has been achieved by using disassembled files, this cannot always be done, particularly if the program uses an unknown packer, and therefore this approach has similar shortcomings to the signature based methods.

Here a similar path is taken to that in Anderson et al. (2011) where they use the dynamic trace from many samples of malware and benign programs to train a classifier. A dynamic trace is a record of the sequence of instructions executed by the program as it is actually running. Dynamic traces can provide much more information about the true functionality of a program than the static binary, since the instructions appear in exactly the order in which they are executed during operation. The drawback to dynamic traces is that the are difficult to collect for two reasons: (i) the program must be run in a safe environment which is more time consuming than processing a static binary file, and (ii) malware often has self-protection mechanisms designed to guard against being watched by a dynamic trace collection tool, and care must be taken to ensure the program is running as it would under normal circumstances.

Here a modified version of the Ether Malware Analysis framework (Dinaburg, Royal, Sharif & Lee 2008) was used to perform the data collection. Ether is a set of extensions on top of the Xen virtual machine. Ether uses a tactic of zero modification to be able to track and analyze a running system. Zero modifications preserves the sterility of the infected system, and limits the methods that malware authors can use to detect if their malware is being analyzed. Increasing the complexity of detection makes for a much more robust analysis system.

Collecting dynamic traces can be slow due to all of the built in functionality of Ether to safeguard against a process altering its behavior while being watched. It is an engineering hurdle to develop a software/hardware solution that would be efficient enough to collect traces on a host without disruption to the user. This problem is being investigated, however, the current implementation is sufficient for a sandbox type on-line application (Goldberg, Wagner, Thomas & Brewer 1996). For example, many institutions implement an email/http inspection system to filter for spam and malware. Inserting the proposed methodology into this process allows for a more robust approach to analyzing new threats in real time.

The three main goals of this work are then to (i) classify malware with high accuracy for a fixed false discovery rate (e.g., 0.001), (ii) provide an online classification scheme that can determine when enough trace data has been collected to make a decision, and (iii) cluster the suspected malware instance with other known malware samples to stream-line the reverse engineering process. Reverse engineering is the process of determining the program's functionality in an effort to better understand the nature and source of the malicious intrusion.

In order to accomplish (i) we propose a logistic regression framework using penalized splines. Estimation of the large number of model parameters is performed with a Relaxed Adaptive Elastic Net procedure, which is a combination of ideas from the Relaxed LASSO (Meinshausen 2007), Adaptive LASSO (Zou 2006), and the Elastic Net (Zou & Hastie 2005). To accomplish (ii) we allow the regression model to have measurement error in the predictors in order to account for the uncertainty in a dynamic trace with a small number of instructions. Lastly (iii) can be accomplished through a novel probability change measure, where the "distance" is based on how much change occurs in the probabilistic surface when moving from one program to another in covariate space. Initial results indicate the potential for vast improvement in detection ability of the proposed method over signature based methods. Further, the proposed clustering strategy makes reverse engineering much more efficient as demonstrated in Section 5.

The rest of the paper is laid out as follows. Section 2 describes the dynamic trace data and how it will be used in the regression model. In Section 3 the underlying classification model and estimation methodology are presented. The classification results on five minute dynamic traces are presented in Section 4. Finally an illustration of how the method could be applied in practice in an online classification and clustering analysis is provided in Section 5. Section 6 concludes the paper.

2 Dynamic Trace Data

As mentioned previously, a modified version of the Ether Malware Analysis framework (Dinaburg et al. 2008) was used to collect the dynamic trace data. A dynamic trace is the sequence of processor instructions called during the execution of a program. This is in contrast to a disassembled binary *static trace* which is the order of instructions as they appear in the binary file. The dynamic trace is believed to be a better measure of the program's behavior since code packers obfuscate functionality from analysis of static traces. Also, the instructions in a dynamic trace are listed in the order that they were actually executed, as opposed to the order they appear in the binary (and many of the instructions that appear in the static trace may never even be executed). There are other data that can be incorporated too (e.g., packer present?, system calls, file name and location, ...). The framework laid out in Section 3 allows for as many data sources or features that one may wish to include.

The point in the dynamic trace where the packer finishes executing instructions is called the original entry point (OEP). At this point in the trace the program will begin to execute instructions related to the actual functionality of the program. Intuitively, it would seem beneficial to remove the portion of the dynamic trace related to the packer. Two data sources are considered (with and without packer instructions, along with a binary predictor to indicate

the presence of packer or not) in the results of Section 4.

In order to make efficient use of the dynamic trace, it is helpful to think of the instruction sequence as a Markov chain. This representation of the sequence has been shown to have better explanatory power than related n-gram methods (Anderson et al. 2011). To this end, the instruction sequence is converted into a transition matrix $\boldsymbol{Z}$ where

$$Z_{jk} = \text{number of direct transitions from instruction } j \text{ to instruction } k.$$

Estimated transition probabilities $\hat{\boldsymbol{P}}$ are obtained from counts $\boldsymbol{Z}$, where

$$P_{jk} = \Pr\{\text{next instruction is } k \mid \text{current instruction is } j\}.$$

The elements of $\hat{\boldsymbol{P}}$ are then used as predictors to classify malicious behavior. This entire procedure is described in more detail in Section 3. The Z_{jk} are 2-grams, while the estimated P_{jk} are essentially a scaled version of the 2-grams, i.e., the relative frequency of going from state$j$ to state k given that the process is now in state j. These quantities (Z_{jk} and P_{jk}) are usually substantially different in this case, since not all states are visited with similar frequencies. Anderson et al. (2011) used the elements of an estimated P_{jk} from dynamic traces (with the state space consisting of Intel instructions observed in the sample) as features in a support vector machine. They found that using the elements of P_{jk} provided far better classification results for malware than using 2-grams (or n-grams in general). This is likely due to the fact that sometimes informative transitions $j \to k$ may occur from a state j that is rarely visited overall, but when it is visited, it tends to produce the $j \to k$ transition prominently. Such situations will be measured very differently with P_{jk} versus Z_{jk}.

There are hundreds of commonly used instructions in the Intel instruction set, and thousands of distinct instructions overall. A several thousand by several thousand matrix of transitions, resulting in millions of predictors would make estimation difficult. Additionally, many instructions perform the same or similar tasks (e.g., 'add' and 'subtract'). Grouping such instructions together in a reasonable way not only produces a faster method but provides better explanatory power vs. using all distinct Intel instructions in our experience.

Through collaboration with code writers familiar with assembly language, we have developed four categorizations of the Intel instructions, ranging from course groupings to more fine.

- Categorization 1 (8 classes → 64 predictors):
  math, *logic*, *priv*, *branch*, *memory*, *stack*, *nop*, *other*
- Categorization 2 (56 classes → 3136 predictors):
  asc, *add*, *and*, *priv*, *bit*, *call*, *math_other*, *movc*, *cmp*, *dcl*, ...
- Categorization 3 (86 classes → 7396 predictors):
  Python Library "pydasm" categories for Intel instructions

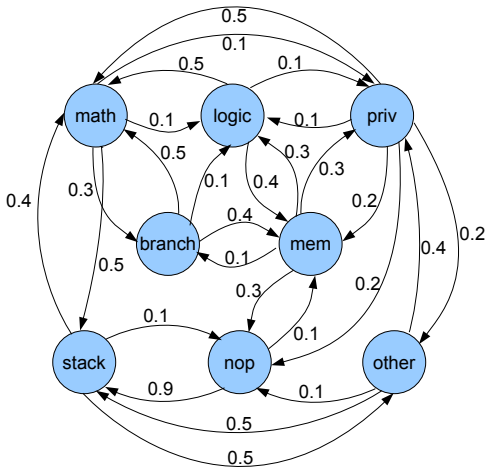
- Categorization 4 (122 classes → 14884 predictors):
  pydasm categories for instructions with *rep instruction-x* given its own class distinct from *instruction-x*.

Figure 1 displays a conceptualization of the Markov Chain Transition Probability Representation of a Dynamic Instruction Trace. The graph on the right has eight nodes corresponding to the eight categories of Categorization 1, where the edges correspond to transition probabilities from one instruction category to the next for the given program. The location each instruction acted on in memory is not used in this analysis, since these locations are not consistent from one execution of the program to another.

The data set used contains dynamic traces from 543 malicious and 339 benign programs, respectively, for a total of 882 observations. The benign sample was obtained from a malware vendor's clean data set, and includes Microsoft Office programs, popular games, etc. The malicious sample was obtained via a random sample of programs from the website `http://www.offensivecomputing.net/`, which is a repository that collects malware instances in conjunction with several institutions. There is a noticeable lack of public sources of malware and malware analysis available. Other malware samples that were available were either for sale or limited to a small number of users. Malware samples are acquired by the Offensive Computing Website through user contributions, capture via mwcollectors and other honey pots, discovery on compromised systems, and sharing with various institutions. Admittedly, this is not a truly random sample from the population of all malicious programs that a given network may see, but it is the largest publicly available malware collection on the Internet (Quist 2012).

Figure 1: Markov chain transition probability representation of a dynamic instruction trace: (left) the first several lines from a dynamic trace output (i.e., instruction and location acted on in memory, which is not used), (right) a conceptual conversion of the instruction sequence into Categorization 1 transition probabilities.

Instr	Address
lea	ecx, [ecx]
sub	esp, 0x3C
sub	esp, 0x3C
mov	ebx, eax
mov	ebp, esp
add	ebx, 0x00410F1F
lea	eax, [ebp+]
mov	[esp+0x14], ecx
sub	eax, ecx
mov	[ebp+], edi
sub	edx, esi
or	edi, 0x0040A24E
xchg	[ebp+], esi

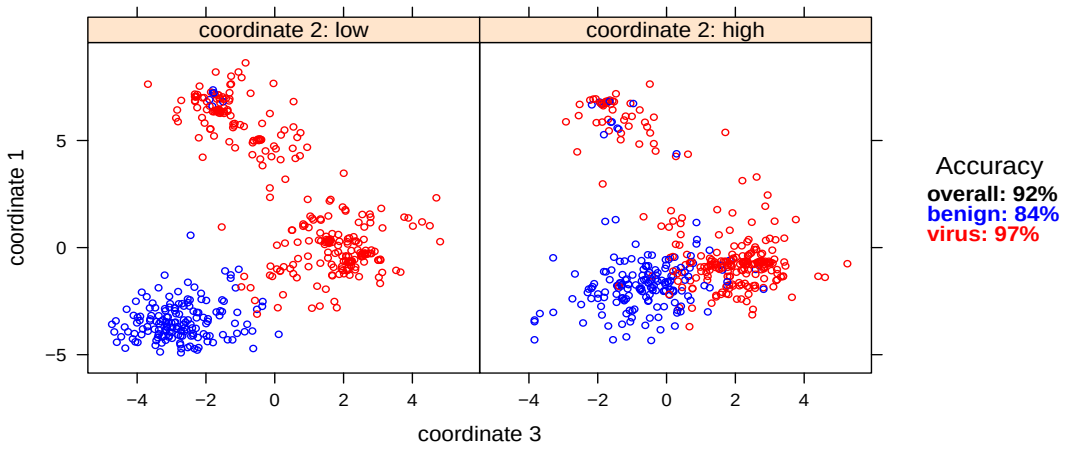


Each of the observations was obtained from a 5 minute run of the program. Originally there were 543 malicious and 473 benign programs, but any program with less than 2000 instructions executed during the five minutes was removed from the data set. The rationale behind this was that some benign processes remain fairly idle, waiting for user interaction. Since such programs produce very short traces and are not representative of the kind of programs that require scanning, it was decided to remove them from the training set. The dataset used in this analysis (dynamic trace files obtained with Ether) is available upon request.

As a first pass at visualizing the data and to get a feeling for how well the malicious samples separate out from the benign samples, a dimension reduction normal mixture model (Hastie & Tibshirani 1996) was fit to the *logit* of the transition probabilities resulting from Categorization 2. For this analysis, the estimated transition probabilities $\hat{\boldsymbol{P}}_i$ for the i-th observation were taken to be the posterior mean (i.e., $\hat{\boldsymbol{P}}_i = \mathrm{E}(\boldsymbol{P}_i \mid \boldsymbol{Z}_i)$), assuming symmetric Dirichlet(0.1) for each row of $\boldsymbol{P}_i$. More details on the estimation of $\boldsymbol{P}_i$ are provided in Section 3. Logistic regression with the elastic net (Zou & Hastie 2005), was first used to screen for *active* predictors among the $56 \times 56 = 3136$ candidate predictors. The R package *mda* was then used to fit the normal mixture model with two components on K linear combinations of the remaining active predictors. The number of linear combinations (dimensions) and their coefficients are estimated along with the mixture parameters as described in (Hastie & Tibshirani 1996). Figure 2 displays a plot of the benign and malicious samples (in blue and red, respectively) on the reduced dimension axes for the resulting three dimensions.

The accuracy numbers reported in Figure 2 are obtained via 10-fold cross validation (CV) of the normal component mixture model. This accuracy ($\sim$ 92% overall) gives a baseline

Figure 2: Normal Component Mixture Data Visualization and Classification: $56 \times 56 = 3136$ predictors for categorization 2, projected to three dimensions that support good cross-validated classification accuracy.



performance metric with which to compare to the procedure described in Section 3.

3 A Statistical Model for Malware Classification

Elements of $\hat{\boldsymbol{P}}$ were used as predictors, similar to the approach used for data visualization in Section 2. Two Main Goals for the classification model are to (i) Screen out most predictors to improve performance and allow certain transitions to demonstrate their importance to classification, and (ii) explicitly account for uncertainty in $\hat{\boldsymbol{P}}$ for online classification purposes because this uncertainty will have a large impact on the decision until a sufficiently long trace is obtained. While (ii) is not as necessary for offline analysis, it can still can be useful in a framework to decide how long to run a program (i.e., how long of a trace is needed in order to make a decision?).

3.1 Logistic Spline Regression Model

For a given categorization (from the four categorizations given in Section 2) with c instruction categories, let $\boldsymbol{Z}_i$ be the transition counts between instruction categories for the i-th observation. Let B_i be the indicator of maliciousness, i.e., $B_i = 1$ if i-th sample is malicious, and $B_i = 0$ otherwise. For the initial model fit discussion in this section, take $\hat{\boldsymbol{P}}_i$ to be the posterior mean (i.e., $\mathrm{E}(\boldsymbol{P}_i \mid \boldsymbol{Z}_i)$), assuming symmetric Dirichlet(λ) for each row of $\boldsymbol{P}_i$. In this analysis $\lambda = 0.1$ was used. In Section 5 a simple approach is described to account for the uncertainty inherent in $\boldsymbol{P}_i$ when making decisions. The assumption is that the training set has observations where the traces are long enough to make the uncertainty in the precise value of $\boldsymbol{P}_i$ somewhat negligible for the purposes of model estimation. This can be verified intuitively through the results of Section 5.1, where probability estimates become fairly precise after about 10,000 instructions.

The actual predictors used to model the B_i are

$$\boldsymbol{x}_i = \left[\mathrm{logit}(\hat{P}_{i,1,1}), \mathrm{logit}(\hat{P}_{i,1,2}), \ldots, \mathrm{logit}(\hat{P}_{i,c,c-1}), \mathrm{logit}(\hat{P}_{i,c,c})\right]', \; i = 1, \ldots, n, \tag{1}$$

where $\hat{P}_{i,j,k}$ is the (j,k)-th entry of the $\hat{\boldsymbol{P}}_i$ matrix, and each component of the $\boldsymbol{x}_i$ is scaled to have sample mean 0 and sample variance 1, across $i = 1, \ldots, n$. The scaling of the predictors to a comparable range is a standard practice for penalized regression methods (Tibshirani 1996, Zou & Hastie 2005). We then use the model

$$\mathrm{logit}[\Pr(B = 1)] \;=\; f_{\boldsymbol{\beta}}(\boldsymbol{x}) \;=\; \beta_0 + \sum_{s=1}^{c^2} \sum_{l=1}^{K+1} \beta_{s,l} \phi_{s,l}(x_s), \tag{2}$$

where the basis functions $\phi_{s,1}, \ldots \phi_{s,K+1}$ form a linear spline with K knots at equally spaced

quantiles of x_s, $s = 1, \ldots, c^2$ (and c^2 is the number of elements in the $\hat{\boldsymbol{P}}$ matrix).

Pairwise products of the $\phi_{s,l}(x)$ can also be included to create a *two-way interaction* spline for $f(\boldsymbol{x})$. A compromise which is more flexible than the additive model in (2), but not as cumbersome as the full two-way interaction spline is to include multiplicative interaction terms into the additive model, i.e.,

$$\text{logit}[\Pr(B=1)] \;=\; f_{\boldsymbol{\beta}}(\boldsymbol{x}) \;=\; \beta_0 + \sum_{s=1}^{c^2}\sum_{l=1}^{K+1} \beta_{s,s,l}\phi_{s,s,k}(x_s) + \sum_{s=1}^{c^2-1}\sum_{t=s+1}^{c^2}\sum_{l=1}^{K+1} \beta_{s,t,l}\phi_{s,t,l}(x_s x_t), \quad (3)$$

where $\phi_{s,t,1}, \ldots \phi_{s,t,K+1}$ form a linear spline with K knots at equally spaced quantiles of $x_s x_t$ for $s \neq t$ (and at equally spaced quantiles of x_s for $s = t$). The model in (3) with $K = 5$ is ultimately the route taken as it produced the best classification results on the problem at hand. This model has potentially a very large number of parameters (βs) in this application, so some efficient sparse estimation procedure is necessary in order to estimate all parameters. This procedure is described next in Section 3.2.

3.2 Relaxed Adaptive Elastic Net Estimation

In order to estimate the large number of parameters in (3), a combination of the Elastic Net (Zou & Hastie 2005), Relaxed LASSO (Meinshausen 2007), and Adaptive LASSO (Zou 2006) procedures was used. The Elastic Net is efficient and useful for extremely high dimensional predictor problems $p >> n$. This is in part because it can ignore many unimportant predictors (i.e., it sets many of the $\beta_{s,t,l} \equiv 0$). The Elastic Net, Relaxed LASSO, and Adaptive LASSO procedures, are reviewed below, then generalized for use in this application.

The data likelihood is

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{n} \left[\text{logit}^{-1}(f_{\boldsymbol{\beta}}(\boldsymbol{x}_i))\right]^{I_{B_i=1}} \left[1 - \text{logit}^{-1}(f_{\boldsymbol{\beta}}(\boldsymbol{x}_i))\right]^{I_{B_i=0}}$$

The Elastic Net estimator is a combination of ridge regression and LASSO (Tibshirani 1996), i.e., the $\boldsymbol{\beta}$ that minimizes

$$\log L(\boldsymbol{\beta}) + \lambda \left\{ \rho \sum_{s=1}^{c^2}\sum_{t=s}^{c^2}\sum_{l=1}^{K+1} \beta_{s,t,l}^2 + (1-\rho) \sum_{s=1}^{c^2}\sum_{t=s}^{c^2}\sum_{l=1}^{K+1} |\beta_{s,t,l}| \right\}, \quad (4)$$

for given tuning parameters $\lambda > 0$ and $\rho \in [0, 1]$ which are typically chosen via m-fold CV. For the linear spline model of (3), the penalty on $\beta_{s,t,l}^2$ and $|\beta_{s,t,l}|$ corresponds to a penalty on the overall trend and the change in slope at the knots (i.e., encourages "smoothness"). The parameters λ and ρ are tuning parameters typically chosen via 10-fold CV, for example. Another

benefit to the Elastic Net is that fits for many values of λ are obtained at the computational cost of a single least squares fit (i.e., $O(p^2)$) via the Least Angle Regression (LARS) algorithm (Efron, Hastie, Johnstone & Tibshirani 2004).

The Relaxed Lasso and Adaptive LASSO both emerged as procedures developed to counteract the over-shrinking that occurs to the nonzero coefficients when using the LASSO procedure in high dimensions. The Relaxed LASSO can be thought of as a two-stage procedure, where the LASSO procedure (i.e., the Elastic Net estimator with $\rho = 0$) is applied with $\lambda = \lambda_1$, then the LASSO is applied again to only the nonzero coefficients with $\lambda = \lambda_2$, where typically $\lambda_1 > \lambda_2$.

The Adaptive LASSO is also a two stage procedure where an initial estimate of $\beta_{s,t,l}$ is obtained via unregularized MLEs or via ridge regression (if $p > n$). In the second step, the LASSO is applied with a penalty that has each term weighted by the reciprocal of the initial estimates, $\tilde{\beta}_{s,t,l}$.

This motivates the following three step approach taken to estimate the coefficients of the logistic spline model in (3).

Algorithm 1: Estimation Procedure.

Step 1: Screen predictors x_s for importance using a linear logistic model

$$f_1(\boldsymbol{x}) = \alpha_0 + \sum_s \alpha_s x_s,$$

with $\boldsymbol{\alpha}$ estimated via Elastic net in (4) with $\lambda = \lambda_1$ and $\rho = .5$.

Step 2: Use active predictors (i.e., those x_s with $\alpha_s \neq 0$) to fit the interaction spline model of (3) via the Elastic Net with $\lambda = \lambda_2$ and $\rho = .5$. Denote, the estimated coefficients from Step 2 as $\tilde{\beta}_{s,t,l}$

Step 3: fit the interaction spline model of (3) via the Adaptive Elastic Net with $\lambda = \lambda_3$ and $\rho = \rho_3$. That is, $\hat{\boldsymbol{\beta}}$ is given by the minimizer of

$$\log L(\boldsymbol{\beta}) + \lambda_3 \left\{ \rho_3 \sum_{s=1}^{c^2} \sum_{t=s}^{c^2} \sum_{l=1}^{K+1} \left(\frac{\beta_{s,t,l}}{\tilde{\beta}_{s,t,l}} \right)^2 + (1-\rho_3) \sum_{s=1}^{c^2} \sum_{t=s}^{c^2} \sum_{l=1}^{K+1} \left| \frac{\beta_{s,t,l}}{\tilde{\beta}_{s,t,l}} \right| \right\}. \qquad (5)$$

The tuning parameters λ_1, λ_2, λ_3, and ρ_3 need to be chosen via cross validation. On the surface it may seem overkill to combine these three concepts, but the extremely high dimensionality of the model in (3) demands this aggressive approach. There are over 9 million parameters if using Categorization 2, over 200 million predictors if using Categorization 4. The initial out-of-sample classification results using just one of these procedures alone were far inferior to those

obtained with combined approach. For example, overall 10-fold CV classification rates of ∼96% were achieved with the Elastic Net, Adaptive LASSO, and Relaxed LASSO, respectively, when used alone to fit the model in (3). Whereas overall 10-fold CV accuracies achieved using the combined method above are ∼99%, as shown in Section 4.

3.3 Prior Correction for Sample Bias

Prior correction (Manski & Lerman 1977, Prentice & Pyke 1979) involves computing the usual logistic regression fit and correcting the estimates based on prior information about the proportion of malware in the population of interest π_1 and the observed proportion of malware in the sample (or sampling probability), $\bar{B}$. Knowledge of π_1 can come from some prior knowledge, such as expert solicitation or previous data. King & Zeng (2001) point out, provided the estimates of the regression coefficients (i.e., $\beta_{s,t,l}$, $j < k, l = 1, \ldots, M$ in (5)) are consistent, then the following corrected estimate is consistent for β_0,

$$\tilde{\beta}_0 = \hat{\beta}_0 - \log\left[\left(\frac{1-\pi_1}{\pi_1}\right)\left(\frac{\bar{B}}{1-\bar{B}}\right)\right]. \tag{6}$$

Prior correction will have no effect on classification accuracy results discussed in Section 4, since it is just a monotonic transformation, so there will be an equivalent threshold to produce the same classifications either way. However, it can be useful in practice to have the estimated probability of maliciousness for a given program provide a measure of belief of the maliciousness of the program on a scale that reflects the appropriate prior probability that the code is malicious. That is, if π_1 can somehow be specified for the given network on which the program will be executed, then prior correction in (6) can be useful.

4 Classification Results

Let $\widehat{\Pr}(B = 1 \mid \boldsymbol{x})$ be given by (3) with $\beta_{s,t,l}$ replaced by their respective estimates $\hat{\beta}_{s,t,l}$. The i^{th} observation is classified as malicious if $\widehat{\Pr}(B = 1 \mid \boldsymbol{x}_i) > \tau$ for some threshold τ which can be selected to produce an acceptable false discovery rate (FDR).

4.1 Out of Sample Accuracy

The classification accuracy of the proposed method is first examined on the various categorizations, with and without the packer removed. If the packer was removed from the trace, then a binary predictor (packer existence or not) was added to the covariate vector $\boldsymbol{x}$. The 10-fold CV overall accuracy results for these covariate scenarios are provided in Table 1. Overall, there is very little difference between the results with or without packer removed, with possibly the exception of Categorization 2 results. It seems that the effect of the packer (which produces

Table 1: Overall out-of-sample accuracy calculated via 10-fold CV by category and packer (removed on not) using logistic spline regression with Relaxed Adaptive Elastic Net estimation. The standard error of the respective accuracy estimates are provided in parentheses.

	Cat 1	Cat 2	Cat 3	Cat 4
w/packer	0.923 (0.009)	0.986 (0.004)	0.991 (0.003)	0.991 (0.003)
w/o packer	0.923 (0.009)	0.993 (0.003)	0.989 (0.004)	0.992 (0.003)

relatively few instructions relative to the remainder of the program) is washed out by the rest of the instructions. However, this could have more of an impact for shorter traces, particularly when collecting traces and analyzing traces early on in real time such as in Section 5. Categorizations 2, 3, and 4 are generally not much different from each other, but they all perform far better than Categorization 1. In the remainder of the results the Categorization 2 data with packer removed is used, since it provides the most parsimonious model among the best performing covariate sets.

The logistic spline regression with Relaxed Adaptive Elastic Net estimation is compared to various other methods using categorization 2 with packer removed in Table 2. The competing methods are (i) linear logistic regression estimated with Elastic Net (i.e., step 1 of Algorithm 1), (ii) a support vector machine provided by the Python package *shogun*, (iii) the mixture discriminant analysis (MDA) routine of Hastie & Tibshirani (1996) (using the R package *mda*) using two components on the set of covariates with nonzero coefficients from the linear logistic

Table 2: Comparison of classification results using various methods. The logistic spline, logistic linear, SVM, and MDA methods use Categorization 2 with packer removed covariates, and had results calculated via 10-fold CV (same 10 folds were used for each method).

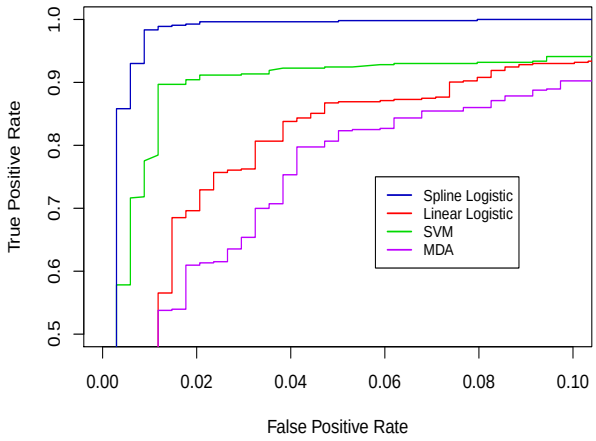
	Overall	Malware Accuracy		
Detection Method	Accuracy	1% FDR[1]	0.3% FDR[2]	∼0% FDR[3]
Spline Logistic (Cat 2)	0.993	0.989	0.858	
Linear Logistic (Cat 2)	0.930	0.564	0.328	
SVM (Cat 2)	0.932	0.862	0.558	
MDA (Cat 2)	0.920	0.538	0.394	
Antivirus 1	0.733			0.632
Antivirus 2	0.537			0.363
Antivirus 3	0.496			0.259

[1] three out of 339 benign programs incorrectly considered malicious
[2] one out of 339 benign programs incorrectly considered malicious
[3] There are *some* false positives from signature-based detection methods due to fuzzy matching heuristics (Antivirus 1 had one false detection in this dataset for example), but the exact FDR for these signature-based methods is unknown.

regression elastic net, and (iv) three leading signature based antivirus programs with all of their most recent updates. The predictor screening used in conjunction with the MDA method is essential in this case in order to avoid the numerical issues with the procedure that occurred when using all predictors. The number of mixture components (two) was chosen to produce the best 10-fold CV accuracy. The names of the leading antivirus programs are not provided due to legal concerns. It is important to recognize that these AV software packages are using signatures (i.e., blacklists) and whitelists as well as heuristics to determine if a program is malicious. The other classification methods in the table are *not* using signatures or white lists, but these could easily be incorporated into these methods in practice which would only improve upon their performance. Even still, the Spline Logistic method performs extremely well on this data set, and could possibly be a promising addition to AV software.

Figure 3 displays the ROC curves for the various competing methods in Table 2. The antivirus programs are excluded since there is no thresholding parameter with which to vary the false positive rate. It is clear from Table 2 and Figure 3 that the spline logistic model with Relaxed Adaptive Elastic Net estimation is far superior to the other methods for this classification problem. In particular, it has an estimated out-of-sample overall error rate of 0.007 (accuracy of 99.3%) which is 10 times smaller than any of the other methods.

Figure 3: ROC Curves for the methods in Table 2



5 Online Analysis of Programs

5.1 Online Detection

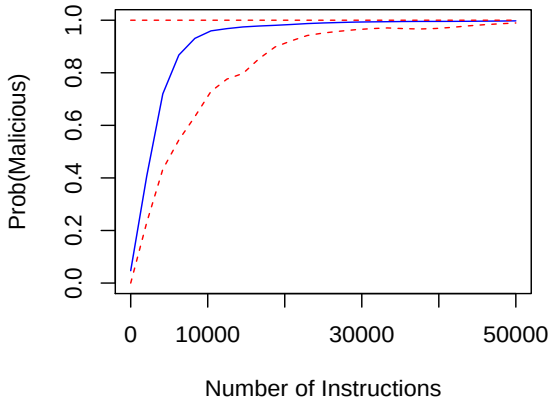
The predictors used in the logit spline model of Section 3 are the elements of the probability transition matrix $\boldsymbol{P}$, which can only be observed (i.e., estimated) with error. This *measurement* error can be substantial for a dynamic trace with only a small number of instructions. For online classification, it is essential to account for this measurement error. Of particular importance is determining how long of an instruction trace is needed before making a decision.

To tackle this issue, the rows of $\boldsymbol{P}$ are further modeled as independent symmetric Dirichlet(λ) vectors *a priori*, which is a conjugate prior for $\boldsymbol{P}$ in a Markov chain model. Thus, for a trace $T_{1:m}$ with m instructions observed thus far, the probability of being Malicious, $\Pr(B = 1) = \text{logit}^{-1}\Big(\hat{f}(\boldsymbol{P})\Big)$, has inherent variability (due to the uncertainty in $\boldsymbol{P}$) that decreases as m in-

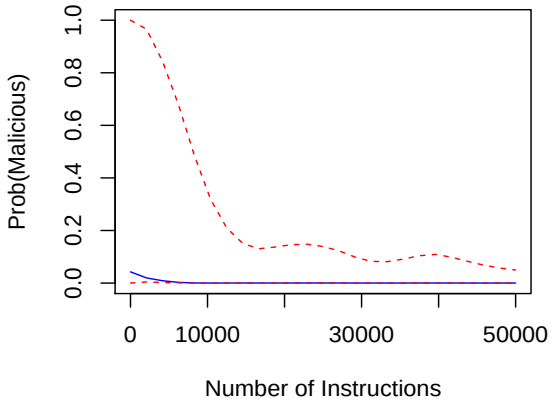
creases (i.e., as a longer instruction trace is obtained). If a given process produces a trace $T_{1:m}$, the distribution of $\Pr(B = 1)$ can be simulated by generating draws from the posterior of $\boldsymbol{P}$ to produce uncertainty bands and a *posterior* mean estimate $\mathrm{E}[\Pr(B = 1) \mid T_{1:m}]$.

This can be thought of as an empirical Bayes approach, as f is replaced with an estimate $\hat{f}$, while only the uncertainty in $\boldsymbol{P}$ is treated. This is a good compromise, as the uncertainty in $\Pr(B = 1)$ is dominated by uncertainty in $\boldsymbol{P}$ early on in the trace. Figure 4 demonstrates this approach on the first malicious and benign processes in the sample, respectively, using a prior correction of $\pi_1 = 0.01$. There is a lot of uncertainty in either case initially, until about 10,000 instructions are collected. By about 30,000 instructions the $\Pr(B = 1)$ for the malicious and benign processes are tightly distributed near one and zero respectively. A possible implementation for online decision making could be to classify as malicious (or benign) according to $\Pr(B = 1) > \tau$ (for some threshold τ that admits a tolerable number of alarms per day) once the 95% credible interval is narrow enough (e.g., < 0.1).

Figure 4: Posterior mean of the probability of malware given the instruction sequence for a malicious sample (a) and benign sample (b), respectively, as a function of number of instructions (95% credible intervals reflecting uncertainty in red)



(a) Dynamic trace from a malicious program



(b) Dynamic trace from a benign program

5.2 Post Detection Analysis

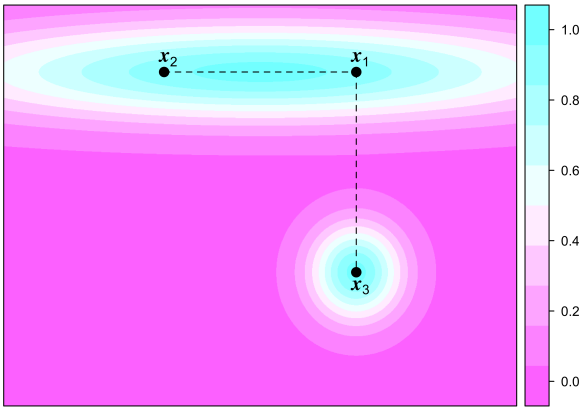
Once a file is suspected to be malicious, it is often necessary to reverse engineer (RE) the program to determine its functionality and origin in order to know how to respond and/or how to better prevent future infections into the computer network. The reverse engineering process is fairly sophisticated, requiring many hours of effort from a highly trained individual. Thus, our goal is to streamline this process by providing useful information about the program. To this end, we describe a novel method to cluster the malicious files. When a new malware instance is detected, it can be clustered into a self-similar group, where perhaps some of the

group members have already been reverse engineered by an analyst. The analyst can then use these previous efforts to more quickly understand the nature and functionality, origin, etc. of the newly suspected malicious program.

There are many clustering approaches in the literature, including hierarchical distance based (Ward 1963, Johnson & Wichern 2007), K-means (MacQueen 1967), Spectral Methods (von Luxburg 2007), and model based procedures (Celeux & Govaert 1995, Hastie & Tibshirani 1996, Fraley & Raftery 2002, Teh, Jordan, Beal & Blei 2006, Hjort, Holmes, Műller & Walker 2010) (see Everitt, Landau & Leese (2001) and Johnson & Wichern (2007) for a review of clustering methods). Here we introduce a similarity measure which takes advantage of the estimated probability $\widehat{\Pr}(B = 1 \mid \boldsymbol{x})$ of being malicious. This similarity measure can then be used in a hierarchical clustering procedure to develop clusters and thus identify neighbors for a given instance of malware.

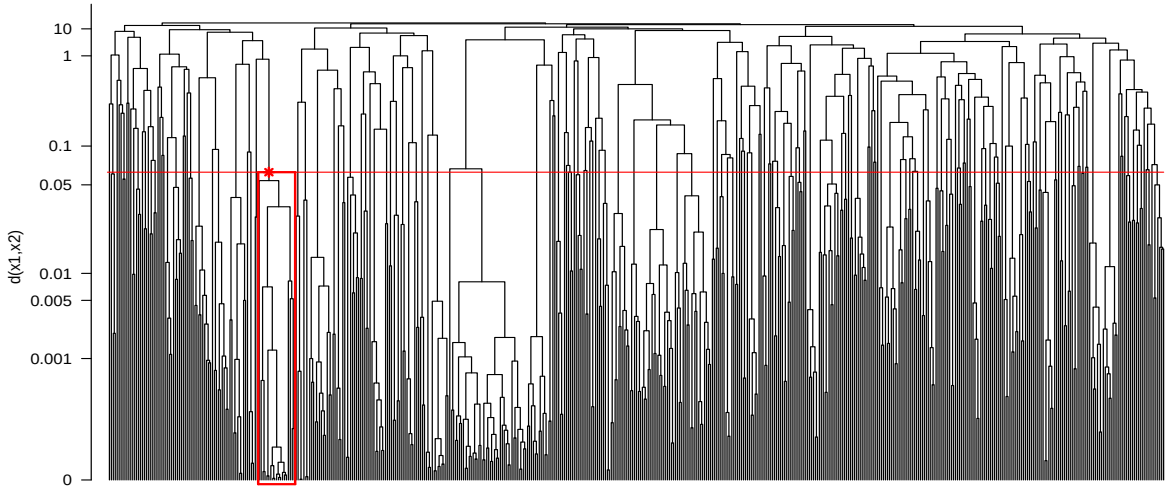
The predictor space in this problem is of very high dimension (3136 predictors for Categorization 2), making direct application of model based clustering methods infeasible. However, as in most problems with high dimensional predictor space, there are relatively few important predictors to the total number. Also, predictors vary substantially in their influence. We postulate that if two observations are very close together with respect to their values of all of the "important" predictors (i.e., those predictors that can inform malware or benign), then they should be considered "neighbors", regardless of their respective values of the other (less informative) predictors. Since the spline logistic model estimation procedure described in Section 3 contains the pertinent information about predictor importance and is useful for classification in this setting, it makes intuitive sense to use this model to define a measure of similarity between observations.

Figure 5: A conceptual probability surface over the predictor space. The Euclidean distance (or Mahalonobis distance if the two predictor dimensions had the same variance) from $\boldsymbol{x}_1$ to $\boldsymbol{x}_2$ is the same as that from $\boldsymbol{x}_1$ to $\boldsymbol{x}_3$. However, the line integral in (7) along the respective dashed lines will be very different, leading to a much larger probability change measure in (7) for $(\boldsymbol{x}_1, \boldsymbol{x}_3)$, than that for $(\boldsymbol{x}_1, \boldsymbol{x}_2)$.



In particular, we define a similarity measure operating on predictor variable space which measures the accumulated change in probability of malware when moving in a straight line from one point ($\boldsymbol{x}_1$) in predictor variable space to another ($\boldsymbol{x}_2$) as depicted in Figure 5. This similarity measure will be much smaller for observations that have little change in the probability surface between them (e.g., $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$), than for observations with much change (e.g., $\boldsymbol{x}_1$ and $\boldsymbol{x}_3$), even

Figure 6: Cluster dendrogram of malware processes using the probability change similarity measure. A horizontal red line is drawn at a cutpoint determined by the fifth percentile of the pairwise distances. The cluster belonging to the malware observation in Figure 4(a) is outlined in red and consists of 19 members.



if there is no difference in Euclidean (or Mahalanobis) distances.

More specifically, the accumulated probability change along the length of the line connecting points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ in Euclidean space can be represented as the line integral

$$d(\boldsymbol{x}_1, \boldsymbol{x}_2) = \|\boldsymbol{x}_1 - \boldsymbol{x}_2\| \left\{ \int_0^1 \left| \frac{\partial}{\partial \lambda} \widehat{\Pr}\Big(B = 1 \,\Big|\, \boldsymbol{x} = \lambda \boldsymbol{x}_1 + (1-\lambda)\boldsymbol{x}_2\Big) \right|^{\rho} d\lambda \right\}^{1/\rho}, \tag{7}$$

where $\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|$ is the standard Euclidean norm. In the analysis presented below $\rho = 1$ is used. The probability change in (7) can be efficiently calculated with a simple quadrature approximation across λ on the consecutive differences of $\widehat{\Pr}(B = 1 \mid \boldsymbol{x} = \lambda \boldsymbol{x}_1 + (1-\lambda)\boldsymbol{x}_2)$.

Figure 6 shows the dendrogram resulting from using Ward's hierarchical clustering method with the similarity measure in (7) to cluster the 543 malicious observations. For reference a horizontal red line is drawn at a cutpoint determined by the fifth percentile of the pairwise distances. In practice, a program will be classified as malicious (such as that in Figure 4(a)), and what is then needed are a few of it's closest neighbors that have been reverse engineered previously to aid in the reverse engineering of the newly suspected malicious program. However, it is possible that the suspected malware has no close neighbors. The fifth percentile of the distances can be used as a rule of thumb to decide if the neighboring observations are close enough to be potentially useful to the analyst. According to this cutpoint, the cluster belonging to the first malware observation in Figure 4(a) is outlined in red and consists of 19 members.

A dynamic trace can be visualized with the Visualization of Executables for Reversing and

Analysis (VERA) software (Quist & Liebrock 2009) in a manner that aids in the reverse engineering process. VERA begins by generating traces logging the address of each instruction. These addresses are then used to form the vertices of the graph. Any observed transition from one address to another generates an edge between the two vertices. Multiple executions of the same transitions between addresses will result in a darker line, indicating a loop. The resulting graph is then arranged with the Open Graph Display Framework Fast Multipole Layout Algorithm (Quist & Liebrock 2009), which generates the graphs seen in Figure 7. Figures 7(a) and 7(b) provide VERA graphs of the suspected malicious program from Figure 4 and its nearest neighbor according to the probability change measure, respectively. An analyst can then observe the graphs and make out structures in them. Similar programs, as denoted in Figure 7, will result in similar graph features.

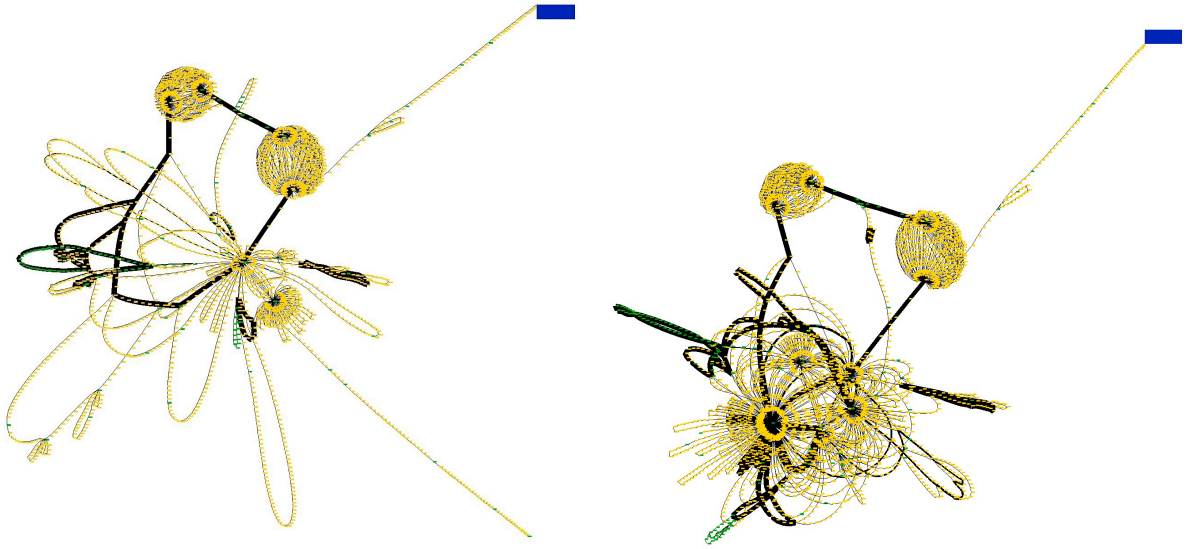
The blue rectangle in the upper right hand corner of both graphs in Figure 7 is the starting address of execution. The instructions proceeding from the upper-right to lower-left are the initialization areas of the loop. Windows executables have initialization code, or preamble, appended to each program. This is referred to as the "initterm" by the Windows compilers. After the initterm code executes, two spheres can be observed in the graph. In the case of the two samples tested, the colocation across two samples is indicative of a common feature set. Moving into cluster of address in the bottom part of the program one can see other similarities between the two samples. Specifically loops and switch statements are similarly located in the graphs. The two graphs are not exactly similar due a generational difference, verified by manual reverse engineering, between the two samples. Given that both samples possess the same execution features, the two samples both share a similar code base and thus are related.

6 Conclusions and Further Work

Flexible classification must be done cautiously with so many predictors. The Relaxed Adaptive Elastic Net is a good framework for adding flexibility with splines. It avoids over-fitting to obtain superior accuracy relative to other popular approaches for classifying new programs. It is also possible to use a model based classification approach that treats the dynamic trace as a Markov Chain directly, and assumes a mixture of Dirichlet distributions for the rows of the transition matrix $\boldsymbol{P}$. This framework would also cluster malware samples as it detects them, and is a subject of further work. Also, (Anderson, Quist, Brown, Storlie & Lane 2012) use additional features (e.g., static trace, file entropy, system calls) to perform classification. The approach discussed here would easily allow for these additional features along with the Ether traces used here for future use.

The current framework allows for online application in a sandbox at the perimeter of a net-

Figure 7: Functionality plots of the dynamic trace obtained with the VERA software of the suspected malware observation in Figure 4(a) and its nearest neighbor according to the probability change measure.



(a) VERA trace plot of the the suspected malware observation in Figure 4(a)

(b) VERA trace plot of the closest neighbor to the the suspected malware observation in Figure 4(a)

work. Implementation of the classification procedure in this manner is currently being pursued for LANL's network. The classification procedure runs very quickly on a given trace (once the model is estimated which can be done offline). The largest obstacle to producing a host-based software (i.e., running the classification procedure on an actual user's machine as it runs the applications) is the collection of dynamic traces in real time efficiently without disruption to the user. The feasibility of such a collection procedure is currently being investigated. Another issue with host based implementation is that users produce instructions traces for benign programs that rely on user interaction, and are thus different from traces produced if the program is left alone. Care must be taken to collect a representative sample of the traces that will be observed on a user's system. Finally, it would be a beneficial extension of this work to incorporate change point detection in order to allow for instances where a benign process is "hijacked" by a malicious program (Cova, Kruegel & Vigna 2010).

Acknowledgments

The authors thank Joshua Neil and Mike Fisk for their helpful discussions related to this paper.

References

Anderson, B., Quist, D., Brown, N., Storlie, C. & Lane, T. (2012), 'Integrating multiple data sources for improved malware classification', *USENIX Security (submitted)* .

Anderson, B., Quist, D., Neil, J., Storlie, C. & Lane, T. (2011), 'Graph-Based Malware Detection using Dynamic Analysis', *Journal in Computer Virology* **7**, 247–258.

Antivirus Comparatives (2011), 'Retrospective test (static detection of new/unknown malicious software)', *Online Report* .

Bayer, U., Moser, A., Kruegel, C. & Kirda, E. (2006), 'Dynamic Analysis of Malicious Code', *Journal in Computer Virology* **2**, 67–77.

Bilar, D. (2007), 'Opcodes as Predictor for Malware', *International Journal of Electronic Security and Digital Forensics* **1**, 156–168.

Celeux, G. & Govaert, G. (1995), 'Gaussian parsimonious clustering models', *Pattern Recognition* **28**, 781 – 793.

Christodorescu, M. & Jha, S. (2003), Static Analysis of Executables to Detect Malicious Patterns, *in* 'Proceedings of the 12th USENIX Security Symposium', pp. 169–186.

Cova, M., Kruegel, C. & Vigna, G. (2010), Detection and analysis of drive-by-download attacks and malicious javascript code, *in* 'Proceedings of the 19th international conference on World wide web', WWW '10, ACM, New York, NY, USA, pp. 281–290.

Dai, J., Guha, R. & Lee, J. (2009), 'Efficient Virus Detection Using Dynamic Instruction Sequences', *Journal of Computers* **4**(5).

Dinaburg, A., Royal, P., Sharif, M. & Lee, W. (2008), Ether: Malware Analysis Via Hardware Virtualization Extensions, *in* 'Proceedings of the 15th ACM Conference on Computer and Communications Security', pp. 51–62.

Efron, B., Hastie, T., Johnstone, I. & Tibshirani, R. (2004), 'Least angle regression', *Annals of Statistics* **32**, 407–499.

Everitt, B., Landau, S. & Leese, M. (2001), *Cluster Analysis*, 4th edn, London: Hodder-Arnold.

Fraley, C. & Raftery, A. (2002), 'Model-based clustering, discriminant analysis, and density estimation', *Journal of the American Statistical Association* **97**, 611–631.

Goldberg, I., Wagner, D., Thomas, R. & Brewer, E. (1996), 'A secure environment for untrusted helper applications (confining the wily hacker)', *Proceedings of the Sixth USENIX UNIX Security Symposium* .

Hastie, T. & Tibshirani, R. (1996), 'Discriminant analysis by gaussian mixtures', *Journal of the Royal Statistical Society. Series B (Methodological)* **58**(1), pp. 155–176.

Hjort, N., Holmes, C., Műller, P. & Walker, S. (2010), *Bayesian Nonparametrics*, Cambridge, UK: Cambridge University Press.

Hofmeyr, S. A., Forrest, S. & Somayaji, A. (1998), 'Intrusion Detection Using Sequences of System Calls', *Journal of Computer Security* **6**(3), 151–180.

Johnson, R. & Wichern, D. (2007), *Applied Multivariate Statistical Analysis*, 6th edn, Upper Saddle River, NJ: Pearson Prentice Hall.

King, G. & Zeng, L. (2001), 'Logistic regression in rare events data', *Political Analysis* **9**(2), 137–163.

Kolter, J. Z. & Maloof, M. A. (2006), 'Learning to Detect and Classify Malicious Executables in the Wild', *The Journal of Machine Learning Research* **7**, 2721–2744.

MacQueen, J. (1967), 'Some methods for classification and analysis of multivariate observations', *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability* **1**, 281–297.

Manski, C. & Lerman, S. (1977), 'The estimation of choice probabilities from choice based samples', *Econometrica* **45**, 1977–1988.

Meinshausen, N. (2007), 'Relaxed lasso', *Computational Statistics and Data Analysis* **52**(1), 374 – 393.

Organisation for Economic Co-operation and Development (2008), 'Malicious software (malware): A security threat to the internet economy', White Paper.
URL: *http://www.oecd.org/dataoecd/53/34/40724457.pdf*

Perdisci, R., Dagon, D., Fogla, P. & Sharif, M. (2006), Misleading Worm Signature Generators Using Deliberate Noise Injection, *in* 'In Proceedings of the 2006 IEEE Symposium on Security and Privacy', pp. 17–31.

Prentice, R. & Pyke, R. (1979), 'Logistic disease incidence models and case-control studies', *Biometrika* **66**, 403–411.

Quist, D. (2012), 'Community malicious code research and analysis'.
URL: *http://www.offensivecomputing.net/*

Quist, D. & Liebrock, L. (2009), Visualizing compiled executables for malware analysis, *in* 'Proceedings of the 6th International Workshop on Visualization for Cyber Security (VizSec 2009)', VizSec, pp. 27–32.

Reddy, D. K. S., Dash, S. & Pujari, A. (2006), New Malicious Code Detection Using Variable Length n-grams, *in* 'Information Systems Security', Vol. 4332 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 276–288.

Reddy, D. & Pujari, A. (2006), 'N-gram analysis for computer virus detection', *Journal in Computer Virology* **2**, 231–239.

Rieck, K., Trinius, P., Willems, C. & Holz, T. (2011), 'Automatic Analysis of Malware Behavior Using Machine Learning', *Journal of Computer Security* **19**(4), 639–668.

Royal, P., Halpin, M., Dagon, D., Edmonds, R. & Lee, W. (2006), Polyunpack: Automating the hidden-code extraction of unpackexecuting malware, *in* 'ACSAC', pp. 289–300.

Shankarapani, M., Ramamoorthy, S., Movva, R. & Mukkamala, S. (2010), 'Malware Detection Using Assembly and API Call Sequences', *Journal in Computer Virology* **7**(2), 1–13.

Stolfo, S. J., Wang, K. & Li, W.-J. (2005), Fileprint Analysis for Malware Detection, *in* 'ACM Workshop on Recurring/Rapid Malcode'.

Stolfo, S., Wang, K. & Li, W.-J. (2007), Towards Stealthy Malware Detection, *in* 'Malware Detection', Vol. 27 of *Advances in Information Security*, Springer US, pp. 231–249.

Symantec (2008), 'Internet Security Threat Report, trends for julydecember 2007 (executive summary)', White Paper.
URL: *http://eval.symantec.com/mktginfo/enterprise/white_papers/b-whitepaper_exec_summary_internet_security_threat_report_xiii_04-2008.en-us.pdf*

Symantec (2011), 'Internet Security Threat Report, Volume 16', White Paper.
URL: *http://www.symantec.com/business/threatreport/index.jsp*

Teh, Y. W., Jordan, M. I., Beal, M. J. & Blei, D. M. (2006), 'Hierarchical dirichlet processes', *Journal of the American Statistical Association* **101**.

Tibshirani, R. (1996), 'Regression shrinkage and selection via the lasso', *Journal of the Royal Statistical Society B* **58**, 267–288.

von Luxburg, U. (2007), 'A Tutorial on Spectral Clustering', *Statistics and Computing* **17**, 395–416. 10.1007/s11222-007-9033-z.
URL: *http://dx.doi.org/10.1007/s11222-007-9033-z*

Ward, J. H. (1963), 'Hierarchical grouping to optimize an objective function', *Journal of the American Statistical Association* **58**(301), 236–244.

Zou, H. (2006), 'The adaptive lasso and its oracle properties', *Journal of the American Statistical Association* **101**, 1418–1429.

Zou, H. & Hastie, T. (2005), 'Regularization and variable selection via the elastic net', *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* **67**(2), 301–320.